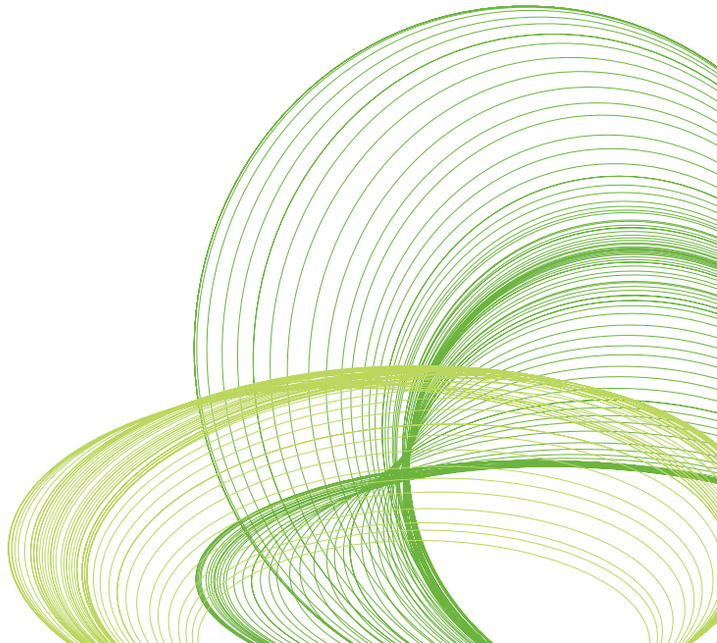
QlikView

QLIKVIEW ARCHITECTURE AND SYSTEM RESOURCE USAGE

QlikView Technical Brief



April 2011

www.qlikview.com

Introduction

This technical brief covers an overview of the QlikView product components and architecture and provides a technical discussion on how QlikView utilizes system resources such as CPU and RAM.

The first section, QlikView Architecture, provides an understanding of the product components and how they fit together to constitute a typical deployment scenario.

The second section, QlikView System Resource Usage describes how QlikView utilizes and interacts with server hardware resources, explains QlikView's approach to data compression and discusses how the different QlikView components use different system resources.

This technical brief is a companion piece to the QlikView Scalability Overview Technology White Paper as it provides a fundamental grounding from which a better understanding of how QlikView scales can be gathered. It is recommended that the reader download and read the Scalability Overview Technology White Paper after reading this technical brief.

QlikView Architecture

When approaching a decision to implement and deploy QlikView, it's important to first understand the roles of the various products that comprise a QlikView deployment.

Figure 1 depicts a simplified view of a standard QlikView deployment containing the location of the various QlikView products as well as both data and application locations.

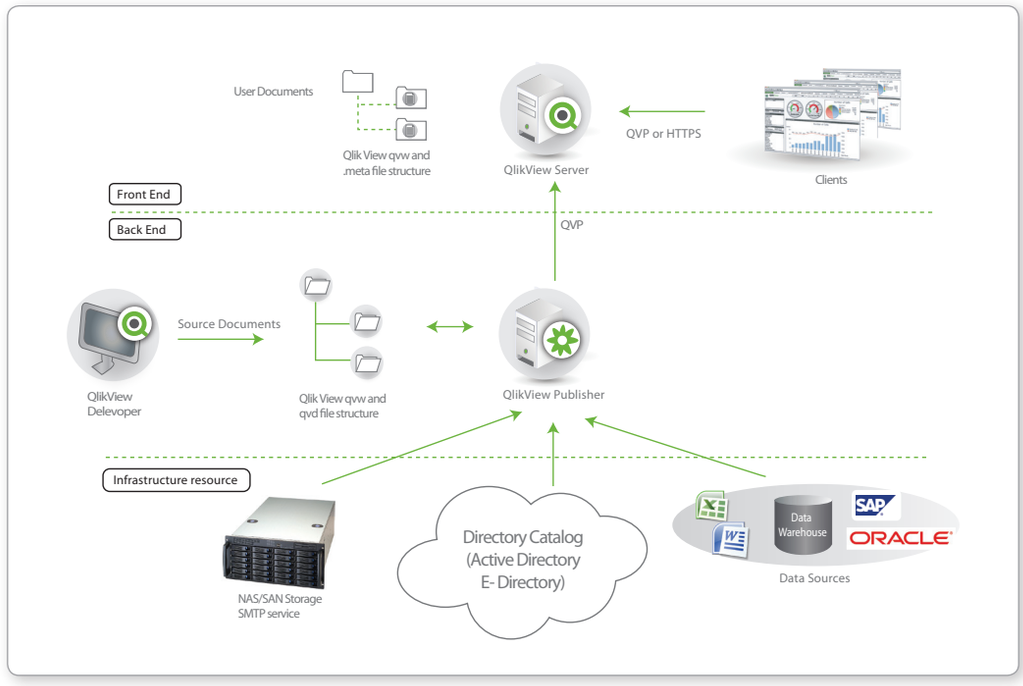


Figure 1: *Architecture Overview.*

QlikView deployments have three main infrastructure components: QlikView Developer, QlikView Server (QVS) and QlikView Publisher.

QlikView Developer is a Windows-based desktop tool that is used by designers and developers to create 1) a data extract and transformation model and 2) to create the graphical user interface (or presentation layer).

QlikView Server (QVS) handles the communication between clients and the QlikView applications. It loads QlikView applications into memory and calculates and presents user selections in real time.

QlikView Publisher loads data from different data sources (oledb/odbc, xml, xls), reduces the QlikView application and distributes to a QVS.

Because QlikView Server and Publisher have different roles and handle CPU and memory differently it's considered a best practice to separate these two components on different servers.

Back End (Including Infrastructure Resources):

This is where QlikView source documents, created using the QlikView Developer, reside. These source files contain either a) scripts within QVW files to extract data from the various data sources (e.g. data warehouses, Excel files, SAP, Salesforce.com) or b) the actual binary data extracts themselves within QVD files. The main QlikView product component that resides on the Back End is the QlikView Publisher: the Publisher is responsible for data loads and distribution. Within the Back End, the Windows file system is always in charge of authorization (i.e. QlikView is not responsible for access privileges).

The Back End depicted in figure 1 is suitable for both development, testing and deployment environments.

Front End:

The Front End is where end users interact with the documents and data that they are authorized to see via the QlikView Server. It contains the QlikView user documents that have been created via the QlikView Publisher on the back end. The file types seen on the Front End are QVW, .meta and .shared documents. All communication between the client and server occurs here and is handled either via HTTPS (in the case of the AJAX client) or via the QlikView proprietary QVP protocol (in the case of the plugin or Windows client). Within the Front End, the QVS is responsible for client security.

Associative In-Memory Technology:

QlikView uses an associative in-memory technology to allow users to analyze and process data very quickly. Unique entries are only stored once in-memory: everything else are pointers to the parent data. That's why QlikView is faster and stores more data in memory than traditional cubes. Memory and CPU sizing is very important for QlikView, end user experience is directly connected to the hardware QlikView is running on. The main performance factors are data model complexity, amount of unique data, UI design and concurrent users.

QlikView System Resource Usage:

At this point it's important to describe at a fundamental level how QlikView's core technology uses system resources like RAM, CPU capacity and so on.
Let's take a look at how both the QlikView Server and the QlikView Publisher both typically use different system resources:

QLIKVIEW SERVER (QVS)

CPU
QlikView Server is multi threaded and optimized to take advantage of multiple processor cores. All available cores will be used almost linearly when calculating the QlikView objects (tables and graphs). The QVS makes a short burst of intense CPU usage when doing any calculations and these are done in real time.

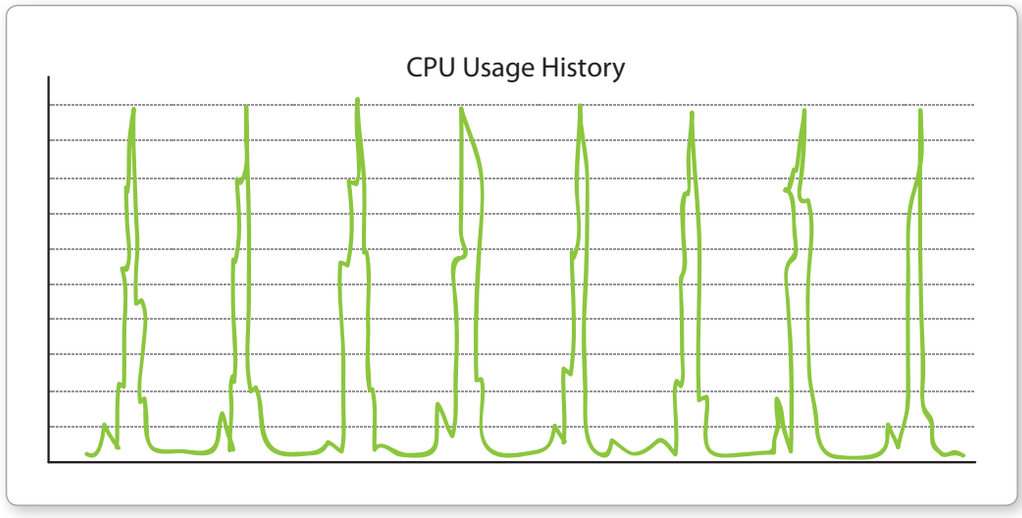


Figure 2: *Typical CPU usage for the calculation of a QlikView Object.*

QlikView Server has a central cache function. This means that QlikView object calculations only need to be done once. Obviously the benefits are better user experience (i.e. faster response times) and lower CPU utilization.

HOW DOES QLIKVIEW USE THE PROCESSOR:

QlikView leverages the processor to dynamically create aggregations as needed in real time resulting in a fast, flexible, and intuitive experience for end users.

It is important to realize that the data stored in RAM is the unaggregated granular data. Typically no preaggregation is preformed in the data reloading/script execution process. When the user interface requires aggregates (e.g. to show a chart object or to recalculate after a selection is made) the aggregation is done in real time. This requires processing power from the CPU.

IMPACT OF NOT ENOUGH PROCESSING POWER:

The primary symptom of a lack of CPU processing power is to wait for charts to recalculate. Under normal conditions chart recalculation takes place almost instantaneously. However with truly massive datasets and without a corresponding increase in processing power, the time to calculate charts can become greater than 1 sec.

If at any time QlikView performance is judged to slow down it is addressed by adding processing power. Quite simply QlikView scales almost perfectly with the addition of more cores and more CPU's. If a given query takes 6 seconds to run against a single core CPU (of a given speed), then the same query will take ~3 seconds to run against a dual core CPU (of the same speed). It will take ~1.5 seconds against a quad core CPU and ~0.75 seconds against two quad core CPUs, etc. One must take into account some additional processing overhead when scaling with cores, however the effects on proportional linear scaling are minimal. Conversely, if additional users are making the same query then the response time will scale linearly according to the number of simultaneous users making the request and the amount of processing power available to the application.

As an increasing number of users make requests to the application with a finite number of cores or CPU's, performance degradation naturally occurs. This is most commonly offset by scaling horizontally using a clustering and load balancing technique.

A major function of the QVS is to load QlikView applications (.qvw's) into memory. The memory size needed depends on:

- QlikView application size (in uncompressed format).
- The application size in memory, it is often bigger than the actual application size.
- How the application is designed. A poorly designed application could utilize unnecessary memory amounts (This topic is covered in the QlikView Scalability Overview White Paper).
- How the data model is designed (e.g. avoiding using synthetic keys can reduce the memory footprint needed).
- Number of users accessing applications on the server (This topic is covered in the QlikView Scalability Overview White Paper).
- A useful rule of thumb is to add 10% extra memory for each additional user: this extra memory is for user state and caching. The cache memory will be reused if needed.

MEMORY:

Main memory RAM is the primary storage location for all data to be analyzed by QlikView.

QlikView uses RAM to store the unaggregated dataset to be analyzed as well as the aggregated data and session state for each user viewing a QlikView document.

QlikView is a snapshot based technology. The snapshot is refreshed through a process known as reloading a QlikView document. When a QlikView document is reloaded QlikView will establish connectivity to the datasource (or datasources) to be analyzed and extract all the *unaggregated granular* data from the data source and then compresses this data. The unaggregated compressed dataset is then saved to disk for persistent storage as a .QVW file.

At the beginning of an analytic session QlikView will load a QlikView document from persistent disk based storage (i.e. a QVW file from hard disk) and place the entire dataset into RAM. During an analytic session QlikView will not make a call out to the database or access any other disk based data repository: It will only rely on the dataset present in RAM. This is what gives QlikView the unlimited flexibility and near instantaneous response times (all data is aggregated in RAM). But, of course, to take advantage of the benefits QlikView provides, all data to be analyzed must fit in RAM.

FACTORS CONTRIBUTING TO QLIKVIEW USAGE OF RAM:

RAM is the single biggest factor determining the quantity of data that can be analyzed in a QlikView environment. There are, however, many factors that determine how much RAM the analysis of a given dataset will require.

The illustration below is a simplified diagram of some of the various usages of RAM that would be found on a typical QlikView Server.

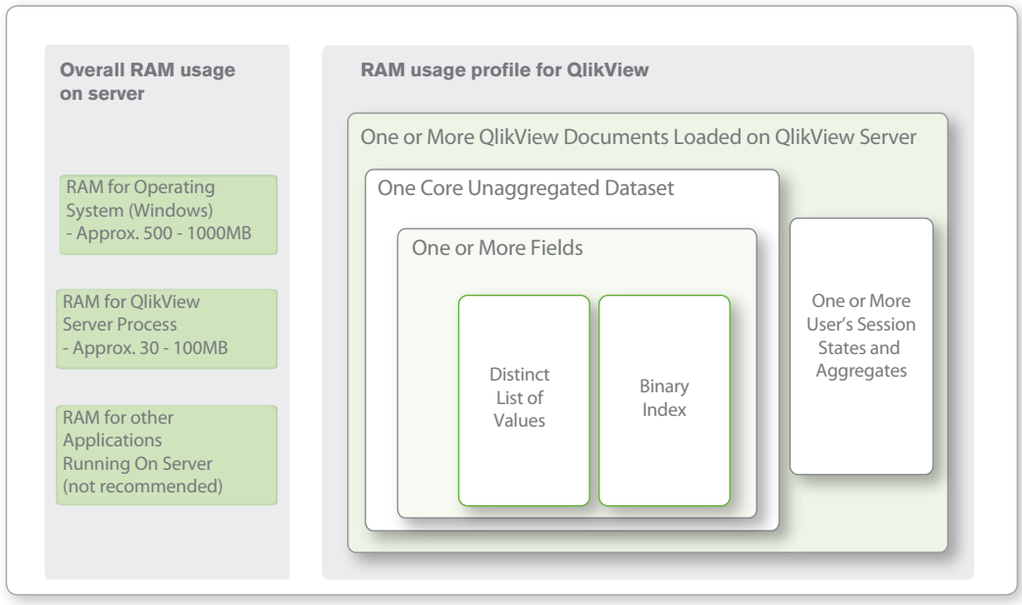


Figure 3: *Memory usage of a QlikView deployment*

RAM FOR THE OPERATING SYSTEM:

A good rule of thumb is to assume a Windows Server Operating System will typically take up 500 to 1000 MB of RAM.

RAM FOR THE QLIKVIEW SERVER PROCESS:

QVS.exe takes up relatively little RAM with no users or documents loaded it will typically be at around 30MB. When documents are loaded and users are connected, QVS.exe will take up more RAM due to the overhead of administering these documents and connections. This is administrative overhead and is separate from the RAM used to load the document itself (i.e. it does not vary with the dataset size). A good rule of thumb is to assume the QVS.exe process will take up 100 MB of RAM.

RAM FOR OTHER APPLICATIONS RUNNING ON THE SERVER:

Running other applications on the same box as the QlikView Server is never recommended. As a general rule the goal is to maximize the amount of RAM that will be available to analyze data in QlikView and running other applications on the same server is contrary to this goal.

One possible exception to this is the running of a web server (either QlikView's HTTP server or Microsoft IIS) on the same machine as the QlikView server purely for convenience. In this scenario the web server should be tasked only with serving QlikView content and not be tasked with running intranet or external web content.

LOADING THE CORE UNAGGREGATED DATASET:

The core unaggregated data set is extracted and compressed during the QlikView reload process. When a file is to be analyzed this core dataset must be loaded into RAM. This dataset is loaded a single time and is not duplicated for multiple users concurrently accessing and analyzing a single document.

It is important to note that it is the characteristics of the data as it is loaded into QlikView that is important not the characteristics of the data as it exists in the original source database. QlikView scripting offers an extremely robust ETL capability that can either increase or decrease the memory required depending on the characteristics of the final dataset produced by the ETL process and ultimately loaded into QlikView.

DATA COMPRESSION: THE UNIQUENESS OF THE DATA IN EACH FIELD LOADED

Almost universally people want to start the QlikView RAM usage discussion with the number of records in a database. However, this is not the most important factor in QlikView RAM usage. The most important factor in QlikView RAM usage is the number of distinct data points in a given field not the number of records.

For example suppose there are two fields that contain the following values:

Order Customer
Joe's Pizza
Joe's Pizza
Joe's Pizza
Joe's Pizza
Joe's Pizza
Joe's Pizza
Joe's Pizza
Joe's Pizza
Joe's Pizza
Joe's Pizza

Order Customer
Joe's Pizza
Tom's Dinner
Jim's Pizzeria
Sal's Italian
Liz's Restaurant
Leroy's Place
Jill's Pizza
Jenny's Place
Terry's Diner
Kel's Pizzeria

Figure 4

Loading the first field into QlikView will consume approximately 1/10th the amount of RAM that loading the second field will consume. In extreme cases (like the example above) this can prove to be an order of magnitude or more difference in RAM usage between loading two fields with the exact same number of records.

This pattern of RAM usage is due to the fact that when QlikView is compressing the data during the reload process, QlikView stores the each distinct data point once only and does not store duplicate values.

THE LENGTH OF DATA IN EACH FIELD LOADED:

"Joe's Pizza" will take up less RAM than will an entry with a very long text string. This is done record by record, regardless of how the field is defined by the developer.

THE NUMBER OF RECORDS IN EACH FIELD LOADED:

If QlikView stores the distinct value only once it still must maintain the relationship back to the original instances. Looking at the first field in the example above, if QlikView stores "Joe's Pizza" just once it still needs to store a reference back to the original ten records. This is done by means of storing a binary index for each field. Additional RAM is taken up to store this binary index. The more records in the field (regardless of uniqueness) the lager this index will be. The index is normally quite small but with large numbers of records and large numbers of fields and tables the memory required increases.

THE NUMBER OF DATASETS LOADED:

Each QlikView Document (.QVW file) represents a discrete dataset. Loading a document loads that document's dataset into RAM. As a result, when multiple, separate documents are opened, it means that multiple, separate datasets are loaded into RAM.

THE USER SESSION STATES, AGGREGATES AND UI DRAWING:

When a user opens a QlikView document the Core Unaggregated Dataset gets loaded in to RAM but in order to draw the user interface QlikView must create and store whatever aggregates are defined by the user interface.

For example, reference the following User Interface chart below:

Salesperson	Total Sales	Cost	Margin	Margin %
Tina	$900	$600	$300	33%
Tom	$600	$200	$400	66%
Teresa	$1000	$500	$500	50%

Figure 5

In order to render this chart, QlikView must first access the Core Unaggregated Dataset and calculate these totals and store them before the chart can be drawn on screen. Storing the User Session States and Aggregates takes up RAM above and beyond the RAM used to store the Core Unaggregated Dataset. Each user needs to have his or her own User Session States; Aggregates are shared across all uses in a central cache.

A general rule of thumb is used for estimating the per-user additional overhead associated with new concurrent users is to add between 1% and 10% of RAM above that used by the first user.

For example: A 1GB .qvw document uses around 4GB in RAM for the first user (based on a multiplier of 4x to establish the initial RAM footprint based on file size). This multiplier is normally between 2x and 10x. User number two may increase this by around 10% as their calculations get cached resulting in a required RAM footprint of 4.4GB. User number 3 requires a further 10%, increasing the footprint to 4.8GB, and so on.

In summary, a properly designed QlikView application will not take up more than 1% to 10% of the RAM usage for each additional user after the first.

IMPACT OF TOO LITTLE RAM AVAILABLE:

Like all Microsoft Windows applications, QlikView is dependent on Windows to allocate RAM for QlikView to use. QlikView Server will attempt to reserve RAM when it starts based on the "Working Set Limits" set in the QlikView Server Management Console. If at any time RAM becomes scarce, Windows may, at its discretion, swap some of QlikView's memory from physical RAM to Virtual Memory (i.e. use the hard disk based cache to in place of RAM).

When QlikView is allocated Virtual Memory it may be orders of magnitude slower than when using 100% RAM. This is always an undesirable condition in QlikView and will provide a poorer experience for the end users and may be perceived as an error condition by end users.

It is critical to realize that the process described above holds true for every Windows based application and is not unique to QlikView.

In this respect hardware sizing is nothing new at all and in some respect must be conducted for every machine (laptop, desktop or server) provisioned across the entire enterprise. But, because with QlikView the amount of RAM available will dictate the amount of data that can be analyzed the hardware sizing process is typically one of the first activities in a QlikView deployment, but is certainly no more important than hardware sizing for any other Windows based software application. This topic is covered in the QlikView Scalability Overview Technology White Paper.

Hard Drive:

Because QlikView Server is used to store and process end-user applications that have been generated from a QlikView Publisher, and because these end-user applications are typically much smaller than their source parent applications, large disk space is typically not required for a QVS. Minimum recommended requirements are 75GB HD in a raid 1 configuration.

QLIKVIEW PUBLISHER

CPU:

QlikView Publisher is a database load engine. Every database connection will creates one thread, meaning that for every data load one core will be utilized almost 100%. Therefore, the maximum number of *simultaneous* database loads is usually the same as number of processor cores available. A comparison of how Publisher and the QVS uses CPU resources highlights the best practice of not having both Publisher and QVS on the same server.

HARD DRIVE:

In a well designed system, Publisher will run specially crafted QlikView applications whose only purpose is to create QlikView data files (qvd), QlikView data marts (qvw files with no graphical interface) and/or reduced QlikView end-user documents (qvw files). This creates historical data repositories that QlikView end user applications will load from (a data cache set). The advantage of this is that it reduces database communication and shortens the reload time. The drawback is the disk space needed to store these source files. The QlikView Publisher server often needs

more hard disk space than QVS. Of course, the amount of disk size needed depends on data amount loaded from the source databases. It is recommended to use a raid 5 or SAN/NAS drive with at least 150GB of space.

MEMORY:

Because the Publisher is a database reload engine and file distribution service rather than an analytics engine, it is not as memory intensive as the QVS. Therefore, memory considerations are typically not a key factor in determining server sizing for Publisher instances.